Internalizing Representation Independence with Univalence

Carlo Angiuli[1] Evan Cavallo[1,2] Anders Mörtberg[2] Max Zeuner[2]

February 25, 2021 — Homotopy Type Theory Electronic Seminar Talk

[1]Carnegie Mellon University

[2]Stockholm University

Today

Recently appeared at POPL 2021 (`dl.acm.org/doi/10.1145/3434293`).

- Two motivations, both related to formalization
- Background on cubical type theory + SIP
- Our relational spin on the SIP, formalized in Cubical Agda

(PL-minded folk: see `youtu.be/ZiZGuOqaq9s`)

Representation independence

"Type structure is a syntactic discipline for enforcing levels of abstraction."

—John C. Reynolds [1983]

"One purpose of type checking in programming languages is to guarantee a degree of 'representation independence:' programs should not depend on the way stacks are represented, only on the behavior of stacks with respect to push and pop operations."

—John C. Mitchell [1986]

A tale of two queues

```
record Queue : Type where
  constructor queue
  field
    Q : Type
    empty : Q
    enqueue : ℕ → Q → Q
    dequeue : Q → Maybe (Q × ℕ)
```

A tale of two queues

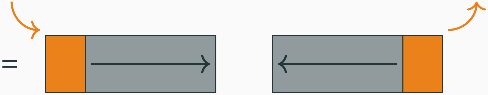
```
record Queue : Type where
  constructor queue
  field
    Q : Type
    empty : Q
    enqueue : ℕ → Q → Q
    dequeue : Q → Maybe (Q × ℕ)
```

```
data List (A : Type) : Type where
  [] : List A
  _::_ : (x : A) (xs : List A) → List A

ListQueue = queue (List ℕ) [] _::_ last
```

A tale of two queues

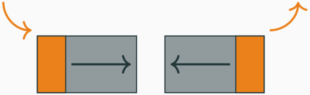
ListQueue = 

BatchedQueue = 

BatchedQueue .dequeue is amortized constant time! [Okasaki 1999]

These are not isomorphic!

ListQueue



BatchedQueue

[] — ([],[])

[1,0] — ([1,0],[])

[1,0] — ([1],[0])

[1,0] — ([],[0,1])

Representation independence

Theorem: Two implementations of an abstract type are observationally equivalent whenever they are related by a structure-preserving correspondence. [Mitchell 1986]

```
record Queue : Type where
field
  Q : Type
  empty : Q
  enqueue : ℕ → Q → Q
  dequeue : Q → Maybe (Q × ℕ)
```

$\mathsf{A}, \mathsf{B} : \mathsf{Queue}$

$\mathsf{R} \subseteq (\mathsf{A}\ .\mathsf{Q}) \times (\mathsf{B}\ .\mathsf{Q})$

$(\mathsf{A}\ .\mathsf{empty})\ \mathsf{R}\ (\mathsf{B}\ .\mathsf{empty})$

$q\ \mathsf{R}\ q' \implies (\mathsf{A}\ .\mathsf{enqueue}\ x\ q)\ \mathsf{R}\ (\mathsf{B}\ .\mathsf{enqueue}\ x\ q')$

$q\ \mathsf{R}\ q' \implies \mathsf{A}\ .\mathsf{dequeue}\ q = \mathsf{nothing}, \ldots$

…via parametricity

$\Longleftarrow$ parametricity metatheorem, e.g., for System F (STLC + $\forall$). [Reynolds 1983]

Lets us reason about ListQueues but actually use BatchedQueues.

Internalizing parametricity

In dependent type theory, can we obtain consequences of parametricity internally?

- Parametricity translations [Bernardy, Lasson 2011; Bernardy, Jansson, Paterson 2012; Keller, Lasson 2012; Anand, Morrisett 2017]
- Add consequences as axioms [Krishnaswami, Dreyer 2013]
- Internal parametricity [Bernardy, Moulin 2012; Bernardy, Coquand, Moulin 2015] (See also [Bezem, Coquand, Huber 2014].)
- ...in cubical type theory [Cavallo, Harper 2020]

Proof reuse and transfer

Motivated by mechanization of mathematics!

Proof that $\zeta(3)$ is irrational requires bounding elements of a sequence using "computations with integers with about 4160 decimal digits." [Chyzak *et al* 2014]

Proof-oriented	**Computation-oriented**
$\mathbb{N}$	~~binary numbers~~ machine integers
matrices	sparse matrices
lists of coefficients	sparse Horner normal form
matrix multiplication	Strassen's algorithm
polynomial multiplication	Karatsuba's algorithm
$\vdots$	$\vdots$

Proof reuse and transfer

CoqEAL [Cohen, Dénès, Mörtberg 2013]

- Parametricity translation, doesn't handle dependently-typed goals.
  (Technical issues modeling large elimination w/Prop-valued relations.)
- Handles the aforementioned examples and more!

Univalent Parametricity [Tabareau, Tanter, Sozeau 2018]

- Univalence, only handles equivalences.

We bridge the gap using univalence and HITs in Cubical Agda.

Cubical Agda

Cubical Agda

Cubical Agda (`agda --cubical`) since Agda 2.6.0. [Vezzosi, Mörtberg, Abel 2019]

≈ De Morgan / CCHM cubical type theory. [Coquand, Huber, Mörtberg 2019]

Formalization and library at `github.com/agda/cubical`.

Define propositional equality in terms of a primitive interval I with $\mathsf{i0}, \mathsf{i1} : \mathsf{I}$.

(And $\_\wedge\_$, $\_\vee\_$ $: \mathsf{I} \to \mathsf{I} \to \mathsf{I}$ and $\sim\_ : \mathsf{I} \to \mathsf{I}$.)

Path types

Given A : Type, $a_0, a_1 : A$,

$$(a_0 \equiv a_1) = \{\mathsf{I} \to A \mid f(\mathrm{i0}) = a_0 \wedge f(\mathrm{i1}) = a_1\}$$

$\mathrm{refl} : (x : A) \to x \equiv x$
$\mathrm{refl}\ x = \lambda\ \_ \to x$

$\mathrm{funExt} : \{f\ g : A \to B\} \to ((x : A) \to f\ x \equiv g\ x) \to f \equiv g$
$\mathrm{funExt}\ p\ i\ x = p\ x\ i$

Dependent path types

Given $A : \mathsf{I} \to \mathsf{Type}$, $a_0 : A(\mathsf{i0})$, $a_1 : A(\mathsf{i1})$,

$$(\mathsf{PathP}\ A\ a_0\ a_1) = \{(i : \mathsf{I}) \to A(i) \mid f(\mathsf{i0}) = a_0 \wedge f(\mathsf{i1}) = a_1\}$$

Expresses $(x : X) \equiv (y : Y)$ "over" $X \equiv_{\mathsf{Type}} Y$.

```
pairExt : (x y : Σ A B)
  → (Σ[ p ∈ (fst x ≡ fst y) ] (PathP (λ i → B (p i)) (snd x) (snd y))) ≃ (x ≡ y)
pairExt x y = isoToEquiv (iso (λ { (p , q) i → (p i , q i) })
                             (λ p → ((λ i → fst (p i)) , (λ i → snd (p i))))
                             (λ _ → refl) (λ _ → refl))
```

Univalence

Kan operations + Glue types give us a computational justification for:

- $\mathsf{transport} : A \equiv B \to A \simeq B$
- $\mathsf{ua} : A \simeq B \to A \equiv B$
- $\mathsf{ua}\beta : \mathsf{transport}\ (\mathsf{ua}\ f) \equiv f$

Higher inductive types

Eliminators compute on path constructors!

$$\text{data } \|\_\| \ (A : \text{Type}) : \text{Type where}$$
$$\quad |\_| : A \to \| A \|$$
$$\quad \text{squash} : (x\ y : \| A \|) \to \boxed{x \equiv y}$$

$$\text{squash} : (x\ y : \| A \|) \to \mathsf{I} \to \| A \|$$
$$\text{squash}\ x\ y\ \text{i0} = x$$
$$\text{squash}\ x\ y\ \text{i1} = y$$

$$\text{map} : (A \to B) \to \| A \| \to \| B \|$$
$$\text{map}\ f\ |\ x\ |\ =\ |\ f\ x\ |$$
$$\text{map}\ f\ (\text{squash}\ x\ y\ i) = \text{squash}\ (\text{map}\ f\ x)\ (\text{map}\ f\ y)\ i$$

Set quotients

```
data _/_ (A : Type) (R : A → A → Type) : Type where
  [_] : (a : A) → A / R
  eq/ : (a b : A) → R a b → [ a ] ≡ [ b ]
  squash/ : isSet (A / R)
```

If R is an (h-prop-valued) equivalence relation, $[a] \equiv_{A/R} [b] \to R\ a\ b$.

The Structure Identity Principle

Structure Identity Principle

Theorem?: Equalities of structured types $\simeq$ structure-preserving equivalences.

But what is a structure-preserving equivalence? What is a structure?

- *Isomorphism is equality* [Coquand and Danielsson 2013]
- HoTT Book [2013]
- *Displayed categories* [Ahrens and Lumsdaine 2017]
- Introduction to Univalent Foundations of Mathematics with Agda [Escardó 2019]
- *The univalence principle* [Ahrens *et al* 2020/2021]

We closely follow Escardó, with cubical modifications.

Structure Identity Principle

$$\mathsf{Monoid} = \Sigma[\, \underbrace{X \in \mathsf{Type}}_{\text{carrier}} \,]\, \underbrace{(X \times (X \to X \to X) \times \cdots)}_{\text{structure}}$$

Definitions:

- A structure is a function $S : \mathsf{Type} \to \mathsf{Type}$.
- An S-structured type is $A : \Sigma[\, X \in \mathsf{Type} \,]\, (S\ X)$.
- Given $A, B : \Sigma[\, X \in \mathsf{Type} \,]\, (S\ X)$ and $f : \mathsf{fst}\ A \simeq \mathsf{fst}\ B$, define a notion of S-structured equivalence $\iota\ A\ B\ f$.
- ι is univalent if $(\iota\ A\ B\ f) \simeq (\mathsf{PathP}\ (\lambda i \to S\ (\mathsf{ua}\ f\ i))\ (\mathsf{snd}\ A)\ (\mathsf{snd}\ B))$.

Pointed types

The pointed structure $S = \lambda X \to X$, with pointed types $(A, a_0), (B, b_0) : \Sigma[\, X \in \mathsf{Type} \,]\, X$.

Definition: $\iota\ (A\ , a_0)\ (B\ , b_0)\ f = (f\,(a_0) \equiv_B b_0)$.

Lemma: ι is univalent.

Proof. That is, $(\iota\ (A\ , a_0)\ (B\ , b_0)\ f) \simeq (\mathsf{PathP}\ (\mathsf{ua}\ f)\ a_0\ b_0)$. But

$$\begin{aligned} & \mathsf{PathP}\ (\mathsf{ua}\ f)\ a_0\ b_0 \\ \simeq\ & \mathsf{transport}\ (\mathsf{ua}\ f)\ a_0 \equiv_B b_0 \\ \simeq\ & f\,(a_0) \equiv_B b_0 . \end{aligned}$$

Structure Identity Principle

Theorem: The natural notion of S-structured equivalence is univalent for:

$$S\ X, T\ X \coloneqq X \mid \alpha \mid S\ X \times T\ X \mid S\ X \to T\ X \mid \mathsf{Maybe}\ (S\ X)$$

We use reflection to automatically match this grammar:

- AutoEquivStr $(\lambda X \to X \times \ldots) \implies$ notion of structured equivalence
- autoUnivalentStr $(\lambda X \to X \times \ldots) \implies$ univalence

Structure Identity Principle

Corollary (SIP): For these (S, ι), we have

$$(A \equiv_{\Sigma[\, X \in \mathsf{Type}\,]\, S\, X}\ B) \simeq (\Sigma[\, f \in \mathsf{fst}\ A \simeq \mathsf{fst}\ B\,]\ \iota\ A\ B\ f).$$

Proof. By univalence of ι:

$$\begin{aligned}
& A \equiv_{\Sigma[\, X \in \mathsf{Type}\,]\, S\, X}\ B \\
\simeq\ & \Sigma[\, p \in \mathsf{fst}\ A \equiv_{\mathsf{Type}} \mathsf{fst}\ B\,]\,(\mathsf{PathP}\ (\lambda i \rightarrow S\ (p\ i))\ (\mathsf{snd}\ A)\ (\mathsf{snd}\ B)) \\
\simeq\ & \Sigma[\, f \in \mathsf{fst}\ A \simeq \mathsf{fst}\ B\,]\,(\mathsf{PathP}\ (\lambda i \rightarrow S\ (\mathsf{ua}\ f\ i))\ (\mathsf{snd}\ A)\ (\mathsf{snd}\ B)) \\
\simeq\ & \Sigma[\, f \in \mathsf{fst}\ A \simeq \mathsf{fst}\ B\,]\,(\iota\ A\ B\ f).
\end{aligned}$$

Axioms

Given a univalent (S, ι) and $\mathsf{ax} : (\Sigma[\, X \in \mathsf{Type} \,]\; S\; X) \to \mathsf{Type}$, define "$S$-structured types satisfying ax" in the evident way; this is univalent if ax is h-prop-valued.

We obtain the usual algebraic structures thusly:

$$\begin{aligned}
&\mathsf{RawMonoidStructure} = \lambda\, X \to X \times (X \to X \to X)\\
&\mathsf{MonoidAxioms}\ (X , \varepsilon , _\cdot_) = \boxed{(\mathsf{isSet}\ X)}\\
&\qquad\qquad\qquad\qquad\qquad \times (\forall\ x\ y\ z \to x \cdot (y \cdot z) \equiv (x \cdot y) \cdot z)\\
&\qquad\qquad\qquad\qquad\qquad \times (\forall\ x \to (x \cdot \varepsilon \equiv x) \times (\varepsilon \cdot x \equiv x))
\end{aligned}$$

Proof transfer for equivalences

Consider the binary numbers Bin (lists of 0, 1 without trailing 0).

We have $f : \mathbb{N} \simeq \mathsf{Bin}$ and in fact $p : (\mathbb{N}, \mathsf{z}, \_+\_, \dots) \equiv_{\mathsf{Monoid}} (\mathsf{Bin}, [], \_+_{\mathsf{Bin}}\_, \dots)$.

Given $P : \mathsf{Monoid} \to \mathsf{Type}$, we have $\mathsf{transport}\ (\lambda i \to P(p\ i)) : P(\mathbb{N}, \dots) \simeq P(\mathsf{Bin}, \dots)$.

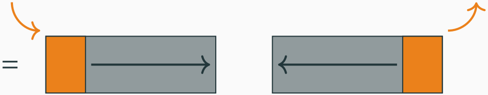
In fact, for $P : \mathbb{N} \to \mathsf{Type}$, we have $P' = \mathsf{transport}\ (\lambda i \to \mathsf{ua}\ f\ i \to \mathsf{Type})\ P : \mathsf{Bin} \to \mathsf{Type}$ and $P(n) \simeq P'(f(n))$, but this implements Bin-addition as $\lambda x\ y \to f(f^{-1}(x) + f^{-1}(y))$.

(So you actually want the SIP here, not just univalence!)

Proof transfer for queues?

ListQueue =

BatchedQueue =



- Not an equivalence of carriers; SIP doesn't apply.
- In fact, BatchedQueues don't even satisfy Queue axioms, e.g., enqueue,dequeue commute on non-empty Queues.

Quotienting BatchedQueues

ListQueue		BatchedQueue
$[]$	—	$([],[])$
		$([1,0],[])$
$[1,0]$	—	$([1],[0])$
		$([],[0,1])$
xs	— $xs \equiv ys \mathbin{++} (\mathsf{reverse}\ ys')$ —	(ys, ys')

A Relational SIP

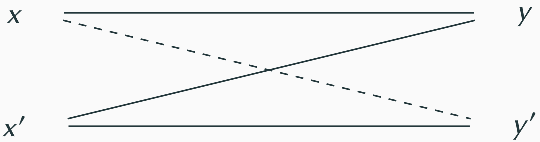
Generalizing queues

Because $R : \mathsf{ListQueue} \to \mathsf{BatchedQueue} \to \mathsf{Type}$ is one-to-many, we can improve it to a (structured) equivalence by quotienting only BatchedQueue.

In general, R is many-to-many. In fact, there may be no (non-HIT) “normal form” representation at all, e.g., finite multisets over a type with no ordering.

Plan: Improve R to a structured equivalence by quotienting both sides. (When possible?)

Quasi–equivalence relations



A relation $R : A \to B \to \mathsf{Type}$ is zigzag-complete if $R\ x\ y \to R\ x'\ y \to R\ x'\ y' \to R\ x\ y'$.
[Tennent and Takeyama 1996; Hofmann 2008; Krishnaswami and Dreyer 2013]

- Graphs of functions are always zigzag-complete.
- Analogue of “symmetry” and “transitivity” for a heterogeneous relation.

Quasi–equivalence relations

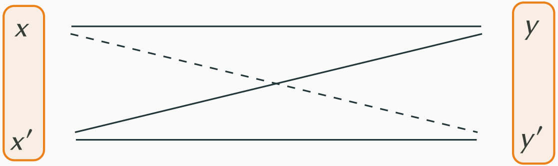
Definition: A relation $R : A \to B \to \mathsf{Type}$ is a QER if it's

- zigzag-complete,
- h-prop-valued,
- $(x : A) \to \| \Sigma[\, y \in B\,]\; R\; x\; y \|$,
- $(y : B) \to \| \Sigma[\, x \in A\,]\; R\; x\; y \|$.

Lemma: Define $R^{\leftarrow} : A \to A \to \mathsf{Type}$ by $R^{\leftarrow}\; x\; x' = \| \Sigma[\, y \in B\,]\; R\; x\; y \times R\; x'\; y \|$.
Then $R^{\leftarrow}, R^{\rightarrow}$ are h-prop-valued equivalence relations $\iff$ R is a QER.

Quasi–equivalence relations

Lemma: Given a QER $R : A \to B \to \mathsf{Type}$, $f : A \,/\, R^{\leftarrow} \simeq B \,/\, R^{\rightarrow}$ where $(f\ [\ x\ ] \equiv [\ y\ ]) \simeq R\ x\ y$. (Proof uses effectivity of quotients.)



Question to audience: Have you seen this somewhere else?

Structured relations

When does a **structured** relation turn into a **structured** equivalence?

Definitions:

- Given $A, B : \Sigma[\, X \in \mathsf{Type}\,]\,(S\ X)$ and $R : \mathsf{fst}\ A \to \mathsf{fst}\ B \to \mathsf{Type}$, define a notion of S-structured relation $\rho\ A\ B\ R$.
- ρ is univalent if some technical conditions hold ("suitable") and it is univalent as a notion of structured *equivalence* on the graphs of equivalences.

A relational Structure Identity Principle

Theorem: The natural notion of S-structured relation is univalent for:

$$S\ X, T\ X := P\ X \mid S\ X \times T\ X \mid \boxed{P\ X} \to S\ X \mid \text{Maybe}\ (S\ X)$$
$$P\ X, Q\ X := X \mid \alpha \mid P\ X \times Q\ X \mid \text{Maybe}\ (P\ X)$$

(Caveats: excludes $\lambda X \to (X \to X) \to X$, and α must be an h-set!)

As before, we use reflection to match this grammar.

Some technical conditions

Definition: A notion of S-structuredness ρ is suitable if the following hold:

- S sends h-sets to h-sets and ρ on h-prop-valued relations is an h-prop.
- If $\rho\ A\ B\ R$ then $\rho\ B\ A\ R^{-1}$.
- If $\rho\ A\ B\ R$ and $\rho\ B\ C\ R'$ then $\rho\ A\ C\ (R \cdot R')$.
- If R is an S-structured h-prop-valued equivalence relation on (X, s), there is a unique S-structure $\overline{s}$ on $X\ /\ R$ for which the graph of $[\_] : X \rightarrow X\ /\ R$ is an S-structured relation between (X, s) and $(X\ /\ R, \overline{s})$.

Some technical conditions

Given S-structures (X, s), (Y, t) and a suitably S-structured QER $R : X \to Y \to \mathsf{Type}$,

$$s \xrightarrow[\text{is } S\text{-structured between}]{} t$$

$$\begin{tikzcd}
X \arrow[r, no head, "R"] \arrow[d, "{[_]}"'] & Y \arrow[d, "{[_]}"] \\
X \,/\, R^{\leftarrow} \arrow[r, dashed, "\simeq"'] & Y \,/\, R^{\rightarrow}
\end{tikzcd}$$

Given S-structures (X, s), (Y, t) and a suitably S-structured QER $R : X \to Y \to \mathsf{Type}$,

$$\begin{tikzcd}
s \arrow[rrr, dash, "\text{is } S\text{-structured between}"'] \arrow[ddd, dash, dashed] & & & t \arrow[ddd, dash, dashed] \\
& X \arrow[r, dash, "R"] \arrow[d, "{[_]}"'] & Y \arrow[d, "{[_]}"] & \\
& X \mathbin{/} R^{\leftarrow} \arrow[r, dashed, "\simeq"'] & Y \mathbin{/} R^{\rightarrow} & \\
\overline{s} \arrow[rrr, dash, dashed] & & & \overline{t}
\end{tikzcd}$$

A relational Structure Identity Principle

Corollary (Relational SIP): S-structured QERs between A, B induce S-structures $A \,/\, R^{\leftarrow} \equiv_{\Sigma[\, X \in \mathsf{Type}\,]\,(S\, X)} B \,/\, R^{\rightarrow}$.

Proof transfer for multisets

$$\text{Multiset } X = X \times \underbrace{(\alpha \to X \to X)}_{\textit{insert}} \times \underbrace{(X \to X \to X)}_{\textit{union}} \times \underbrace{(\alpha \to X \to \mathbb{N})}_{\textit{count}}$$

Given A, B : $\Sigma[\, X \in \text{Type}\,]$ (Multiset X) and a QER R : fst A → fst B → Type, if:

- R A.empty B.empty
- $\forall\ x\ xs\ ys \to$ R $xs\ ys \to$ R (A.insert $x\ xs$) (B.insert $x\ ys$)
- $\forall\ xs\ ys\ xs'\ ys' \to$ R $xs\ ys \to$ R $xs'\ ys' \to$ R (A.union $xs\ xs'$) (B.union $ys\ ys'$)
- $\forall\ x\ xs\ ys \to$ R $xs\ ys \to$ A.count $x\ xs \equiv$ B.count $x\ ys$

Then we immediately obtain equal multiset structures on (fst A) / $R^{\leftarrow}$ and (fst B) / $R^{\rightarrow}$.

Summary

- It's nice to transfer results between structures!
- By the SIP, structured equivalences are equalities of structures. (Currently used in the library's development of $\mathbb{Z}$-cohomology!)
- We establish conditions under which structured relations can be improved to structured equivalences.

All formalized in Cubical Agda: `git.io/JL5x8`.